

Metadata for Energy Disaggregation

Jack Kelly and William Knottenbelt

Department of Computing, Imperial College London, UK.

e-mail: jack.kelly@imperial.ac.uk

Abstract—Energy disaggregation is the process of estimating the energy consumed by individual electrical appliances given only a time series of the whole-home power demand. Energy disaggregation researchers require datasets of the power demand from individual appliances and the whole-home power demand. Multiple such datasets have been released over the last few years but provide metadata in a disparate array of formats including CSV files and plain-text README files. At best, the lack of a standard metadata schema makes it unnecessarily time-consuming to write software to process multiple datasets and, at worse, the lack of a standard means that crucial information is simply absent from some datasets. We propose a metadata schema for representing appliances, meters, buildings, datasets, prior knowledge about appliances and appliance models. The schema is relational and provides a simple but powerful inheritance mechanism.

I. INTRODUCTION

Research suggests that consumers are better able to reduce their energy consumption if given an itemised, appliance-by-appliance energy bill rather than a bill which only describes aggregate consumption [1]. Energy disaggregation is the process of estimating the energy consumed by individual appliances in a home given a time series of the whole-home power demand. A typical use case would be to provide consumers with an estimated itemised electricity bill without requiring the expense of installing separate meters on every appliance.

Research into energy disaggregation (also known as ‘non-intrusive load monitoring’ or NILM) began with George Hart’s pioneering work in 1984 [2], [3]. A recent resurgence has been triggered by a combination of high energy bills and the introduction of ‘smart electricity meters’. In the quest to design and implement a high performance energy disaggregation system, researchers require several types of data:

The primary requirement is for datasets which record the power demand of whole homes as well as the ‘ground truth’ power demand of individual appliances within the home. In 2011, researchers at MIT released the first public dataset for energy disaggregation research [4]. Since then, over ten more datasets have been released and, in March 2014, a project called Wiki Energy¹ launched to share datasets online. These datasets have been received with enthusiasm but different datasets use different file formats. Machine-readable metadata is often minimal and uses a schema and vocabulary unique to that dataset. At best, the lack of a standard metadata schema makes it time-consuming to write software to process multiple datasets. At worse, some datasets simply lack sufficient

metadata to allow the data to be properly interpreted. For example, the mains wiring connecting meters to each other and to appliances forms a tree (with the whole-house meter at the root and appliances at the leaves). In some datasets, this tree structure has more than two levels (i.e. when an appliance turns on or off, the resulting change in power consumption is sensed by more than two meters) yet the metadata rarely specifies the wiring tree.

A secondary requirement is for data describing the behaviour of appliances (e.g. a probability distribution describing the typical times per day that each appliance is used). This prior knowledge can be used to fine-tune the estimates produced by a disaggregation system. Such data is currently available in research papers and industry reports but not in a machine-readable form.

Additionally, it is currently difficult (if not impossible) to directly compare any pair of disaggregation algorithms published in the literature. This is because different researchers tend to use different datasets and different performance metrics. Even if a pair of researchers use the same dataset, they might use different segments of that dataset! If raw disaggregation results and metadata describing the results and training procedures could be published with each research paper then the community could begin to objectively rank the performance of disaggregation algorithms. Such ranking is common in other fields of machine learning such as machine vision. (Ranking energy disaggregation algorithms is a complex task and we certainly do not pretend that the introduction of metadata is sufficient!)

Finally, consumers are unlikely to put effort into training a disaggregation tool. As such, if open-source disaggregation solutions such as NILMTK² are to be viable as consumer-facing disaggregation solutions then researchers must distribute pre-trained models for each appliance. If these models adhere to a standard metadata schema then multiple software systems can exchange models.

Against this background, we propose the first draft of a hierarchical metadata schema for energy disaggregation. Specifically, our schema models electricity meters, appliances (including prior knowledge such as probability distributions describing typical times of use and parameters describing inferred models of appliances), buildings and datasets.

Although we have made every effort to ensure that our proposed schema and controlled vocabularies capture the information present in all the datasets we are aware of, our

¹<http://wiki-energy.org/>

²<http://nilmtk.github.io>

schema can undoubtedly be improved and so the schema are presented as an open-source project³ (under a permissive Apache 2.0 license) to which contributions are most welcome!

In this paper we first outline related work; then we describe the design of our metadata schema; then look at the implementation of a schema validator and finally we discuss conclusions and future work.

II. RELATED WORK

The general principal of using metadata to describe research datasets is not new. For example, the not-for-profit organisation DataCite⁴ (established in 2009) publishes the DataCite Metadata Schema for describing research datasets.

In late-2013, the Nature Publishing Group launched a journal called ‘Scientific Data’⁵ for describing datasets. The machine-readable description of each dataset is captured using the ISA_Tab metadata specification⁶ which specifies a hierarchical schema consisting of the ‘investigation’ (the project context), the ‘study’ (a unit of research) and the ‘assay’ (an analytical measurement).

Biology researchers have embraced the need for metadata schemas and controlled vocabularies as demonstrated by, for example, The Open Biological and Biomedical Ontologies database⁷ which aims to enable the creation of a suite of interoperable reference ontologies for the biomedical domain.

Few metadata projects are specifically for energy datasets. One notable project is Project Haystack⁸ which is an open source initiative to develop taxonomies and tagging conventions for building equipment and operational data. Amongst other achievements, Haystack defines a language for describing electricity meters, including which parameters each meter records and the relationships between meters and between meters and loads. But Haystack is primarily targeted at large commercial buildings rather than domestic buildings, and does not define a controlled vocabulary for appliance names, let alone more granular detail about appliances.

The UK Energy Research Council’s Energy Data Centre provides a simple schema⁹ based on the Dublin Core Metadata Initiative¹⁰ (DCMI).

To summarise the related work: there are many metadata projects for describing research datasets in general but only a small number of metadata projects for describing energy datasets. To the best of our knowledge, there are no existing metadata schemas specifically for describing objects relevant to energy disaggregation. Existing datasets for energy disaggregation do provide some metadata (e.g. a text file mapping appliance names to recording channels) but this metadata does not use a controlled vocabulary and often provides scant details.

³https://github.com/nilmk/nilm_metadata

⁴<http://www.datacite.org>

⁵<http://www.nature.com/scientificdata>

⁶<http://isa-tools.org>

⁷<http://www.obofoundry.org>

⁸<http://project-haystack.org>

⁹<http://ukedc.rl.ac.uk/format.html>

¹⁰<http://dublincore.org>

III. DESIGN

The NILM Metadata schema models several objects relevant to energy disaggregation: electricity meters, appliances, buildings and datasets. The schema specifies property names for each object, the type for each value and controlled vocabularies (e.g. for appliance names and categories). NILM Metadata also provides a database of appliances and meters.

In the sections below, we describe our ‘dataset’ and ‘building’ schemas; the distinction between *meters* and *appliances*; the representation of the mains wiring; the inheritance mechanism for appliances; categorisation; the containment mechanism that allows an appliance to contain other appliances; prior knowledge; and finally our representation of learnt models.

A. Dataset

NILM Metadata models electricity meters, appliances, buildings and datasets arranged in a tree-shaped hierarchy. The dataset object is at the root of this tree.

Most public datasets contain one or more buildings which, in turn, contain one or more meters and one or more appliances. An exception is the ‘tracebase’ dataset [5] which describes appliances and meters without their building context. To accommodate these different scenarios, our dataset object can contain an array of buildings or an ‘electric’ object (a container for meters and appliances).

The dataset schema models properties such as `publication_date`, `rights_list`, `geospatial_coverage`, `temporal_coverage`, `funding`, `creators`, `related_documents`, `timezone` and `geo_location` (the full list of properties and their meaning is given in appendix A).

B. Building

Buildings have an integer ID (unique within the dataset), a list of rooms (using a controlled vocabulary for room names), and some properties shared with dataset: `temporal_coverage`, `geo_location`, `timezone`. These properties default to the values set in the parent dataset but can be overridden per building. Each building contains a `utilities` object which has properties for `electric`, `gas` and `water`. The `electric` object, in turn, has properties for appliances and meters.

C. Meters are distinct from appliances

A tempting simplification would be to assume a one-to-one relationship between electricity meters and appliances. But we often observe one-to-many relationships (e.g. multiple appliances plugged into a multi-way mains adapter which, in turn, is connected to a single meter) and we occasionally observe many-to-one relationships (e.g. in the US and Canada many large domestic appliances like washing machines draw a total of 240 volts from the two 120 volt supplies found in a typical house). We frequently observe situations where some appliances are not submetered. To capture all these scenarios, our schema distinguishes between electrical appliances and meters.

D. Mains wiring

Each building in a typical dataset will have one meter which records the aggregate, whole-building mains power demand. Downstream of this meter might be meters which measure entire circuits within the building (e.g. the lighting circuit). Finally, there are often meters which measure individual appliances.

As such, the mains wiring connecting meters with each other and with appliances can be described as a tree. Each meter can specify either a `submeter_of` property (the numeric ID of the upstream meter) or a `site_meter` property (a boolean flag which is set to `true` if this meter measures the whole-building aggregate). There can be multiple site meters per building; e.g. for three-phase or split-phase mains supplies. The property names ‘`submeter_of`’ and ‘`site_meter`’ are adapted from Project Haystack.

Each appliance is directly downstream of one or more meters so each appliance stores an array of meter IDs (this list will contain only a single meter ID for most appliances except for North American and Canadian dual-supply 240 volt appliances).

E. Appliance inheritance

Electrical appliances can be described as a hierarchical tree of objects. For example, a ‘wine cooler’ can be considered a specialisation of a ‘fridge’ and, as such, inherits properties from fridges.

Inheritance is a well-established technique in software engineering for maximising code re-use. NILM Metadata implements a simple but powerful form of inheritance known as prototype-based inheritance (first implemented in the Self programming language [6] and used in JavaScript). Objects in prototype-based programming languages are not instances of a class but, instead, inherit from any other object (the ‘parent’ or ‘prototype’ object). In NILM Metadata, each appliance object has a ‘parent’ from which it inherits properties. These properties can be modified by the child and the child can specify properties not specified by the parent. The inheritance tree can be any depth.

The end result is that metadata authors only have to specify the parent for each appliance and then NILM Metadata can copy over all prior knowledge about that appliance. Metadata authors are free to override any properties which differ in their appliance.

Inheritance follows a small number of rules. If a property is contained in the parent and absent in the child then it is copied to the child. If a property is present in both parent and child then it is handled differently depending on the type of the property:

- 1) **list (array)** objects become the union of the parent and child lists.
- 2) **scalar** objects in the child override (‘shadow’) properties in the parent.
- 3) **objects** (dictionaries) are recursively updated using the rules above.

Child objects can specify a `do_not_inherit` property (a list of property names) to avoid inheriting named properties.

Subtypes versus inheritance. Appliance objects have a `subtype` property (which must be set to a member of the parent’s `subtypes` set). What is the difference between a subtype and a child object? Subtypes are useful when two related appliances are so similar that we can safely ignore the differences for the purposes of energy disaggregation. For example, an analogue radio and a digital radio are sufficiently similar to mean that they can both be subtypes of the ‘radio’ object. On the other hand, an electric cooker has a significantly different electricity load profile compared to a cooker fuelled by natural gas, so these are separate objects.

Additional properties. Some appliances have rare properties. For example, a television might have a `screen_size` property. We do not want to pollute the common ‘appliance’ schema with these properties (because, for example, it makes no sense for a cooker to be able to specify a `screen_size` property!). Instead, appliance objects can define `additional_properties` to specify the schema for any additional properties (using JSON Schema). `additional_properties` is inherited using the same rules as any other property.

F. Categorisation

When analysing domestic power consumption, we often want to group appliances into certain appliances. For example, we might want to ask ‘what is the total energy consumption for all consumer electronics?’.

Domestic appliances are traditionally classified as one of ‘wet’, ‘cold’, ‘consumer electronics’, ‘ICT’, ‘cooking’, ‘lighting’ or ‘heating’.

An alternative classification is a simple binary classification of ‘large appliances’ (e.g. dish washer) versus ‘small appliances’ (e.g. a radio).

A more finely-grained classification based on the electrical properties of appliances was proposed by Tsagarakis, Colin and Kiprakis 2013 [7]. For example, this taxonomy splits lighting into general incandescent lamps, fluorescent lamps and light-emitting diode (LED) sources. An appliance can have multiple classifications from this taxonomy.

Yet another taxonomy for domestic appliances is the Google product taxonomy¹¹ (used on Google Shopping). This taxonomy is a tree which we represent as list of classifications.

NILM Metadata currently supports all four taxonomies listed above and it would be trivial to add more. We specify a controlled vocabulary for the category names. At present, all appliances in the NILM Metadata object repository have a ‘traditional’ classification and many have classifications for the other taxonomies.

G. Appliances can contain other appliances

Some appliances can be modelled as an aggregation of other objects. For example, a washing machine can be modelled as

¹¹<https://support.google.com/merchants/answer/160081>

a drum motor and a water heating element. Appliance objects in NILM Metadata have a `components` property which is an array of appliance objects. Containment is recursive and can be of any depth.

Of course, *all* appliances can be decomposed into components. Do we model each individual resistor and transistor? No; the end-goal is to model appliances only in sufficient detail to allow an energy disaggregation system to identify the whole appliance given prior knowledge of the components. As such, we only describe individual components if their electrical behaviour is observable from a typical mains electricity meter. It is also important that components be truly separate entities from an electrical perspective. For example, a fridge freezer should *not* be modelled as containing both a fridge and a freezer because that would imply that a fridge freezer has two separate compressors but - as far as we are aware - fridge freezers typically have one compressor.

If an appliance contains multiple instances of the same component then we use the `count` property in the component to specify the number of instances.

The container appliance inherits categories from each of its components. This is useful mostly for the ‘electrical’ taxonomy. For example, if we model a washing machine as a motor and a heater then the washing machine inherits the appropriate electrical classifications from both the motor and the heater.

Our representation of lighting exploits NILM Metadata’s containment mechanism. We distinguish between the light fitting (also called the luminaire or fixture) and the electric lamp(s) within each fitting. We have a ‘light’ object which contains any number of ‘lamps’ (of which there are several kinds including ‘LED lamp’ and ‘incandescent lamp’). Light objects can also contain a ‘dimmer’ object.

H. Prior knowledge

Prior knowledge can be exploited to improve disaggregation performance. Examples of prior knowledge include: the distribution of on-powers of an appliance; the typical time of use per day or per week; correlations with other appliances (e.g. the computer monitor is often on when the computer is on).

NILM Metadata specifies a ‘prior’ object which holds several properties, the two most important of which are `distribution_of_data` (the distribution of the data expressed as normalised frequencies per discrete bin (for continuous variables) or per category (for categorical variables)) and `model` (which describes a model fitted to describe the probability density function (for continuous variables) or the probability mass function (for discrete variables)). We can also specify the `source` of data (is it a subjective guess, or the result of primary data analysis, or taken from a published paper?), whether the prior is `specific_to` a country and what `training_data` was used to generate the prior.

Each ‘appliance’ object has a ‘distributions’ property which is an object with the following properties (each property is an array of priors): `on_power`, `on_duration`,

`off_duration`, `usage_hour_per_day`, `usage_day_per_week`, `usage_month_per_year`, `rooms`, `subtypes`, `appliance_correlations`, `ownership`, `ownership_per_country`, `ownership_per_continent`

We store an *array* of priors (rather than a single prior) for each distribution. This allows us to store multiple beliefs about each distribution (which could be combined using Bayesian statistics). For example, we might find several published papers which provide evidence about the distribution for the power consumption of an appliance. Furthermore, NILM Metadata collects all relevant priors as it descends the inheritance hierarchy for each object (for example, a ‘wine cooler’ might not have any priors associated with it but it will inherit prior knowledge from its parent ‘fridge’ object). Of course, priors from a distant ancestor are less relevant than priors from a recent ancestor so we tag each prior with a `distance` property (a positive integer indicating the number of ‘generations’ away the prior is from the appliance in question).

I. Learnt models of appliances

End-users of domestic disaggregation software are unlikely to put any effort into training the system. This means that we must use either an unsupervised disaggregation algorithm or a supervised learning algorithm with pre-trained models. But note that even if a fully unsupervised disaggregation algorithm is used then the system must still have prior appliance models to be able to provide human-readable names to the appliances. As such, we specify a simple ‘appliance model’. This has properties such as `model_type` (a controlled vocabulary with terms such as ‘HMM’ for hidden Markov model), `training_data`, `date_prepared` etc. The model’s parameters are stored in a `parameters` object.

IV. IMPLEMENTATION

The syntactic elements of the schema are specified using JSON Schema Draft 4¹². The code which implements the semantics of NILM Metadata and performs validation is written in Python. We make use of the `jsonschema`¹³ package for validation and `PyYAML`¹⁴ for loading YAML files.

Prior to validating each appliance, the `properties` object specified by the ‘appliance’ schema is updated with concatenated `additional_properties` specified by the appliance’s ancestors.

Metadata instances can be written in JSON or YAML. We chose JSON/YAML over XML for several reasons: firstly, constructs in JSON/YAML (array, number, string, object, null, boolean) map cleanly onto constructs in standard programming languages whilst XML constructs such as `<x y="z">foo</x>` do not. Also, JSON/YAML is arguably easier for both humans and machines to read than XML and is certainly more compact.

¹²<http://json-schema.org/>

¹³<https://github.com/Julian/jsonschema>

¹⁴<http://pyyaml.org/wiki/PyYAML>

We have written a `process.py` script which takes a hand-written metadata file (in JSON or YAML), concatenates it with the NILM Metadata database, produces a large 'concatenated' metadata file (in either JSON or YAML) and then validates the concatenated metadata against the NILM Metadata schema.

V. CONCLUSIONS

We have proposed the first draft of a metadata schema for representing objects relevant to energy disaggregation. The schema adapts ideas from DCMI, Project Haystack, ISA_Tab and DataCite; and adds new elements relevant to energy disaggregation (only a few of these elements are required to be instantiated). We also propose a simple but powerful inheritance mechanism to minimise duplication of information and effort. The schema has successfully been used to capture metadata for the UK-DALE (Domestic Appliance-Level Electricity) Dataset [8].

Whilst NILM Metadata is fit for use now, there will inevitably be use-cases that we have neglected hence we warmly welcome contributions from the community! NILM Metadata is open-source to facilitate collaboration and is available at github.com/nilmtn/nilm_metadata

VI. FUTURE WORK

We plan to fully integrate the schema and object database with the open source disaggregation toolkit NILMTK [9] and to design a schema for describing disaggregation results using a combination of our existing 'appliance' and 'prior' schemas.

REFERENCES

- [1] C. Fischer, "Feedback on household electricity consumption: a tool for saving energy?" *Energy Efficiency*, vol. 1, no. 1, pp. 79–104, May 2008. [Online]. Available: <http://www.springerlink.com/content/276m42024x61wh1h/>
- [2] G. W. Hart, "Nonintrusive Appliance Load Data Acquisition Method," MIT Energy Laboratory Technical Report, Tech. Rep., Sep. 1984.
- [3] G. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, Dec. 1992.
- [4] J. Z. Kolter, M. J. Johnson, J. Zico Kolter, and Matthew J. Johnson, "REDD: A Public Data Set for Energy Disaggregation Research," in *ACM Special Interest Group on Knowledge Discovery and Data Mining, workshop on Data Mining Applications in Sustainability*, San Diego, CA, USA, 2011. [Online]. Available: <http://redd.csail.mit.edu/kolter-kddsust11.pdf>
- [5] A. Reinhardt, P. Bauman, D. Burgstahler, M. Hollick, H. Chonov, M. Werner, and R. Steinmetz, "On the Accuracy of Appliance Identification Based on Distributed Load Metering Data," in *Proceedings of the 2nd IFIP Conference on Sustainable Internet and ICT for Sustainability*, Pisa, Italy, 2012, pp. 1–9.
- [6] C. Chambers, D. Ungar, and E. Lee, "An efficient implementation of SELF a dynamically-typed object-oriented language based on prototypes," *ACM SIGPLAN Notices*, vol. 24, no. 10, pp. 49–70, Oct. 1989. [Online]. Available: <http://dl.acm.org/citation.cfm?id=74878.74884>
- [7] G. Tsagarakis, A. J. Collin, and A. E. Kiprakis, "Statistical Review of UK Residential Sector Electrical Loads," p. 18, Jun. 2013. [Online]. Available: <http://arxiv.org/abs/1306.0802>
- [8] J. Kelly and W. Knottenbelt, "'UK-DALE': A dataset recording UK Domestic Appliance-Level Electricity demand and whole-house demand," 2014. [Online]. Available: <http://arxiv.org/abs/1404.0284>
- [9] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, "NILMTK: An Open Source Toolkit for Non-intrusive Load Monitoring," in *ACM E-Energy*, Cambridge, UK, 2014.

APPENDIX A SCHEMA PROPERTIES

Dataset

schema (*string*) The URL of the NILM Metadata version (git tag) against which this metadata is validated.

mains_voltage (*object*) Nominal mains voltage. Properties: nominal (e.g. '230'), tolerance_upper_bound, tolerance_lower_bound.

geo_location (*object*) Default geographical location for each building. Properties: country, locality, latitude, longitude.

timezone (*string*) Standard TZ name from the IANA (Olson) TZ database.

data_formats_and_locations (*array*) Each item has properties: format (e.g. 'CSV'), raw_data_location, cleaned_data_location.

funding (*array of strings*) All the sources of funding used to produce this dataset.

description_of_subjects (*string*) Are they all MSc computing students, for example?

item_type (*enum*) One of {buildings, electric}

number_of_building (*int*)

buildings (*array of building objects*)

electric (*electric object*) Stores meters and appliances.

Only one of {buildings, electric} may be used at once.

Elements in 'dataset' schema adapted from standard DCMI elements: **identifier**, **subject**, **creators**, **publisher**, **name (title)**, **publication_date**, **rights_list**, **description**, **geospatial_coverage**, **temporal_coverage**, **related_documents**.

Building

Shares some properties with the 'dataset' schema:

temporal_coverage, **geo_location**, **timezone**.

building_id (*int*) Unique within dataset.

n_occupants (*int*) Max number of occupants.

periods_unoccupied (*array of timeframes*)

rooms (*array of room objects*)

utilities (*electric object*)

Room

name (*string*) Controlled vocabulary of room names.

instance (*int*)

floor (*int*)

Timeframe

start (*string*) Start date in ISO 8601 format.

end (*string*) End date in ISO 8601 format.

Electric

meters (*array of meter objects*)

appliances (*array of appliance objects*)

mains_voltage (*object*) same definition as `dataset:mains_voltage`.

Device

Elements adapted from standard DCMI elements:

creators, **year_of_purchase**,
year_of_manufacture, **description**.

Self-documenting string properties: **name**, **model**,
model_url, **manufacturer**, **seller**, **brand**,
brand_url, **subtype**, **images** (*array of strings*),
part_number, **gtin**, **version**

parent (*string*) Name of prototype object.

dates_active (*array of timeframes*) Used to specify a change in appliance over time (for example if one appliance is replaced with another).

room (*object*) Properties:

name (*string*): room name (e.g. 'kitchen')

instance (*int*): room instance

Meter

Includes all properties from 'device' schema.

meter_id (*int*) Unique identifier within building.

site_meter (*bool*) True if this meter measures the whole building aggregate power demand.

submeter_of (*int*) `meter_id` of upstream meter.

data_location (*string*) Path to data file, relative to dataset's root directory.

sample_period (*float*) Sample period in seconds.

max_sample_period (*float*) Maximum allowed sample period (threshold which defines a 'gap').

measurements (*array of measurement objects*)

wireless (*bool*) True if this meter is wireless.

wireless_base (*device object*) Details of the wireless base station.

data_logger (*device object*) Details of the data logging system.

warning (*string*) Any issue to be aware of.

preprocessing_applied (*array of objects*) Controlled vocab describing which preprocessing steps have already been applied.

gap_free_segments (*array of timeframes*) Segments where sample period never drops below `max_sample_period`.

contiguous_segments (*array of timeframes*) Segments where sample period always equals `sample_period`.

energy (*array of objects*) Each item has properties: `start`, `end`, `active`, `reactive`, `apparent`.

Appliance

Includes all properties from 'device' schema.

meter_ids (*array of ints*) IDs of the electricity meter(s) monitoring this appliance.

categories (*object*) Properties:

traditional: One of {`wet`, `cold`, `consumer electronics`, `ICT`, `cooking`, `heating`}

size: One of {`small`, `large`}

electrical: Any of {`lighting`, `incandescent`, `fluorescent`, `compact`, `linear`, `LED`, `resistive`, `SMPS`, `no PFC`} (this is just a subset).

google_shopping: (*array of strings*)

additional_properties (*object*) Schema specifying additional properties for this type of appliance. Specified using JSON Schema.

subtypes (*array of strings*) Allowed subtypes.

on_power_threshold (*float*) Watts defining threshold between 'on' and 'off'.

control (*array of strings*) Any of {`timer`, `manual`, `motion`, `sunlight`, `thermostat`, `always on`}. For example, a video recorder would be both 'manual' and 'timer'.

efficiency_rating (*object*) Properties:

`certification_name`, `rating`.

nominal_consumption (*object*) Properties (each is a *float*): `on_power`, `standby_power`, `energy_per_year`, `energy_per_cycle`

components (*array of appliance objects*)

usual_components (*array of strings*)

distributions (*object*) Probability distributions.

Properties (each is an *array of priors*): `on_power`, `on_duration`, `off_duration`, `usage_hour_per_day`, `usage_day_per_week`, `usage_month_per_year`, `rooms`, `subtypes`, `appliance_correlations`, `ownership`, `ownership_per_country`, `ownership_per_continent`.

Prior

description, specific_to

distribution_of_data (*object*) Distribution of the data expressed as normalised frequencies per discrete bin (for continuous variables) or per category (for categorical variables). Properties:

bin_edges (*array of floats*)

categories (*array of strings*)

values (*array of floats*)

model (*object*) A fitted model to describe the probability density function (for continuous variables) or the probability mass function (for discrete variables). Use additional properties for the relevant parameters. Properties:

distribution_name (*string*) e.g. 'normal'.

sum_of_squared_error (*float*)

source (*enum*) One of subjective, empirical from data, empirical from publication. If from publication then use related_documents to provide references. If from data then provide details using the software and training_data properties.

specific_to (*object*) Is this prior specific to a particular country or continent? Properties: country, continent.

distance (*int*) The distance (in numbers of generations) between this prior and the most-derived object. In other words, the larger this number, the less specific to the object this prior is. If this is not set the prior applies to the current object.

training_data (*object*) Properties: dataset, buildings, dates

Self-documenting properties: n_datapoints,

date_prepared, related_documents, software

Measurement

physical_quantity (*enum*)

One of {power, energy, voltage}

type (*enum*)

One of {active, reactive, apparent}

maximum (*float*) Maximum sane value.

minimum (*float*) Minimum sane value.

description (*string*) Description of measurement.

Appliance Model

appliance (*string*)

model_type (*enum*) One of HMM, FHMM, combinatorial optimisation

parameters (*object*)

Shares properties with the 'prior' schema:

training_data, software,

related_documents, date_prepared,